VIETNAM NATIONAL UNIVERSITY - HO CHI MINH CITY

UNIVERSITY OF SCIENCE

FACULTY OF INFORMATION TECHNOLOGY

Project 2: Image Processing

Course: Applied Mathematics and Statistics for Information Technology

MTH00057 - 23CLC04

Students:
Dinh Xuan Khuong - 23127398

Instructors:
Nguyen Ngoc Toan
Tran Ha Son

July 22, 2025

Table of Contents

1 Introduction

1.1 About the project

This project focuses on fundamental image processing techniques using Python. It includes:

- Reading and displaying images
- Saving images after manipulation
- Basic image transformation functions:
  - Brightness control
  - Contrast control
  - Flip the image (Vertically and Horizontally)
  - Apply grayscale and sepia filter
  - Sharpen the image
  - Make the image blurred (Box, 3x3, 5x5 kernel)
  - Crop the image in the center
  - Circular crop the image and symmetrical 2-ellipse crop the image

The project uses common libraries such as **numpy**, **matplotlib**, and **PIL**.

1.2 Input and output

- **Input:** A square colored image.
- **Output:** An image after transformation.

1.3 Objective

To develop a basic image processing application capable of reading, transforming, and saving images using various functions such as brightness and contrast adjustment, flipping, filtering, sharpening, blurring, and cropping.

2 Methods

2.1 Helper functions

2.1.1 read_img(img_path)

Input:

- `img_path` (`str`): Path to the input image file.

Output:

- RGB image as a NumPy array (`np.ndarray` with shape [H, W, 3]).

Functionality:

Reads an image from the given file path using PIL, converts it to RGB format, and returns it as a NumPy array.

2.1.2 show_img(img)

Input:

- `img` (`np.ndarray`): Image array to be displayed.

Output:

- No return value; displays the image using `matplotlib`.

Functionality:

Displays an image without axis ticks using `matplotlib.pyplot.imshow()`.

2.1.3 save_img(img, img_path)

Input:

- `img` (`np.ndarray`): Image to be saved.
- `img_path` (`str`): Destination path for saving the image.

Output:

- No return value; saves the image in PNG format at the given path.

Functionality:

Converts a NumPy image array back to a PIL Image and saves it using the specified path and format.

2.2 Image Enhancement Functions

2.2.1 Brightness Control

Idea:

Brightness is adjusted by adding a constant value α to every pixel in the image. Increasing α makes the image brighter, while decreasing it darkens the image. The resulting values are clipped to the valid range $[0, 255]$:

$$\text{New Value} = \min(\max(R/G/B + \alpha,\ 0),\ 255)$$

Input:

- `img` (`np.ndarray`): Input image (2D or 3D).
- `alpha` (`int`): Brightness adjustment factor with range from -255 to 255 and default value is 55.

Output:

- Brightness-adjusted image (`np.ndarray`).

Functionality:

Adjust the brightness of the image.

2.2.2 Contrast Control

Idea:

Contrast is adjusted by scaling the difference between each pixel and the midpoint intensity value (128). Higher α values increase contrast; lower values reduce it:

$$\text{New Value} = \min(\max((R/G/B - 128) \times \alpha + 128,\ 0),\ 255)$$

Why minus 128:

- In 8-bit images, pixel intensities range from 0 to 255.
- The mid-gray point is 128.
- When we multiply pixel values directly by alpha, we scale both dark and bright areas away from black, possibly losing the midtone balance.
- By subtracting 128, we're effectively centering the contrast adjustment around the midtone, making the transformation symmetric with respect to gray.

Input:

- `img` (`np.ndarray`): Input image (2D or 3D).
- `alpha` (`float`): Contrast scaling factor.

Output:

- Contrast-adjusted image (`np.ndarray`).

Functionality:

Centers pixel values around 128, scales by α, then re-centers and clips to the range $[0, 255]$. This changes the intensity range to enhance or reduce contrast.

2.2.3 Image Flipping

Idea:

The idea behind flipping is to swap pixels along the chosen axis. For a given image of size $n \times n$, this can be achieved by iterating from the edges inward:

$$\text{For } i = 0 \text{ to } \left\lfloor \frac{n}{2} \right\rfloor : \begin{cases} \text{Swap rows } i \text{ and } (n-i-1), & \text{for vertical flip} \\ \text{Swap columns } i \text{ and } (n-i-1), & \text{for horizontal flip} \end{cases}$$

Input:

- `img` (`np.ndarray`): Input image.
- `orientation` (`str`): 'V' for vertical flip, 'H' for horizontal (default is 'V').

Output:

- Flipped image (`np.ndarray`).

Functionality:

Flip an image along a specified axis.

2.3 Color Filters

2.3.1 Grayscale Conversion

Idea:

Convert an RGB image to grayscale by applying a weighted sum of the Red, Green, and Blue channels. The weights are based on human visual perception:

$$\text{Gray} = 0.299 \cdot R + 0.587 \cdot G + 0.114 \cdot B$$

The result is replicated across all three channels for consistency in visualization.

Input:

- `img` (`np.ndarray`): Input RGB image of shape `[height, width, 3]`.

Output:

- Grayscale image with three identical channels (`np.ndarray`).

Functionality:

Convert the original image to a grayscale.

2.3.2 Sepia Conversion

Idea:

Apply a sepia filter by linearly transforming RGB values to give the image a warm, antique look. This is done using the following transformation:

$$\begin{bmatrix} R' \\ G' \\ B' \end{bmatrix} = \begin{bmatrix} 0.393 & 0.769 & 0.189 \\ 0.349 & 0.686 & 0.168 \\ 0.272 & 0.534 & 0.131 \end{bmatrix} \cdot \begin{bmatrix} R \\ G \\ B \end{bmatrix}$$

Input:

- `img` (`np.ndarray`): Input RGB image of shape `[height, width, 3]`.

Output:

- Sepia-toned image (`np.ndarray`) of the same shape.

Functionality:

Apply a sepia filter to the image.

2.4 Image Convolution

2.4.1 Sharpening

Idea:

Sharpening enhances image edges and details by applying a high-pass convolution kernel. It highlights intensity differences between a pixel and its neighbors:

$$\text{Kernel} = \begin{bmatrix} 0 & -1 & 0 \\ -1 & 5 & -1 \\ 0 & -1 & 0 \end{bmatrix}$$

Input:

- `img` (`np.ndarray`): RGB image with shape `[H, W, 3]`.

Output:

- Sharpened image of the same shape (`np.ndarray`).

Functionality:

Sharpen the image by padding the image with it edges, applyping the sharpening kernel to each color channel using nested loops.

2.4.2 Box Blur

Idea:

Box blur smooths the image by averaging the pixels in a 3×3 neighborhood using a uniform kernel:

$$\text{Kernel} = \frac{1}{9} \begin{bmatrix} 1 & 1 & 1 \\ 1 & 1 & 1 \\ 1 & 1 & 1 \end{bmatrix}$$

Input:

- `img` (`np.ndarray`): RGB image with shape `[H, W, 3]`.

Output:

- Blurred image (`np.ndarray`) of same shape.

Functionality:

Make the image blurred by padding the image with zeros and performing 2D convolution with the box kernel over each RGB channel.

Why Choose Box Blur:

- Better performance on simple averaging tasks
- Time-efficient due to fast convolution
- Test on 700x700 image with runtime:
  - Box kernel: 13.8325 (seconds)
  - Gaussian 3x3 kernel: 13.8876 (seconds)
  - Gaussian 5x5 kernel: 14.3818 (seconds)

Test results:

Box kernel

Gaussian 3x3 kernel

Gaussian 5x5 kernel

More about Gaussian blur:

Work same as box blur but with different kernels.
The 3×3 kernel uses:

$$\text{Kernel} = \frac{1}{16}\begin{bmatrix} 1 & 2 & 1 \\ 2 & 4 & 2 \\ 1 & 2 & 1 \end{bmatrix}$$

The 5×5 kernel uses:

$$\text{Kernel} = \frac{1}{256}\begin{bmatrix} 1 & 4 & 6 & 4 & 1 \\ 4 & 16 & 24 & 16 & 4 \\ 6 & 24 & 36 & 24 & 6 \\ 4 & 16 & 24 & 16 & 4 \\ 1 & 4 & 6 & 4 & 1 \end{bmatrix}$$

2.5 Crop Functions

2.5.1 Center Crop

Idea:

Extracts the center region of the image by keeping the middle 50% in both dimensions:

$$\text{Crop} = \left[\frac{H}{4} : \frac{3H}{4},\ \frac{W}{4} : \frac{3W}{4}\right]$$

Input:

- `img` (`np.ndarray`): RGB image with shape `[H, W, 3]`.

Output:

- Cropped image of shape `[H/2, W/2, 3]` (`np.ndarray`).

Functionality:

Computes the center height and width, slices the image accordingly, and returns the cropped region. And this is **the image is cropped in the center with its size equal to 1/4 original size**.

2.5.2 Circular Crop

Idea:

A circular mask is applied to retain only the central circular region of the image. All pixels outside the circle are set to black. The radius is chosen as the smallest distance from the image center to its edge.

Input:

- `img` (`np.ndarray`): RGB image with shape `[height, width, 3]`.

Output:

- Circularly masked image (`np.ndarray`) of the same shape.

Functionality:

Generates a boolean mask for pixels inside a circle centered at the image midpoint. The original image is copied to a black canvas where only the circular region is retained.

2.5.3 Elliptical Crop (Dual Symmetrical)

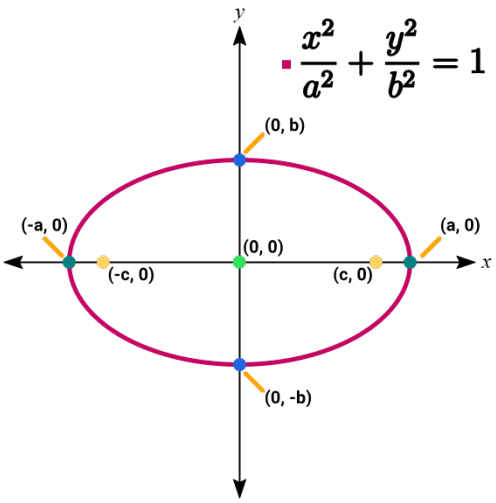
Hard part we need to tackle

How can we define a and b in:

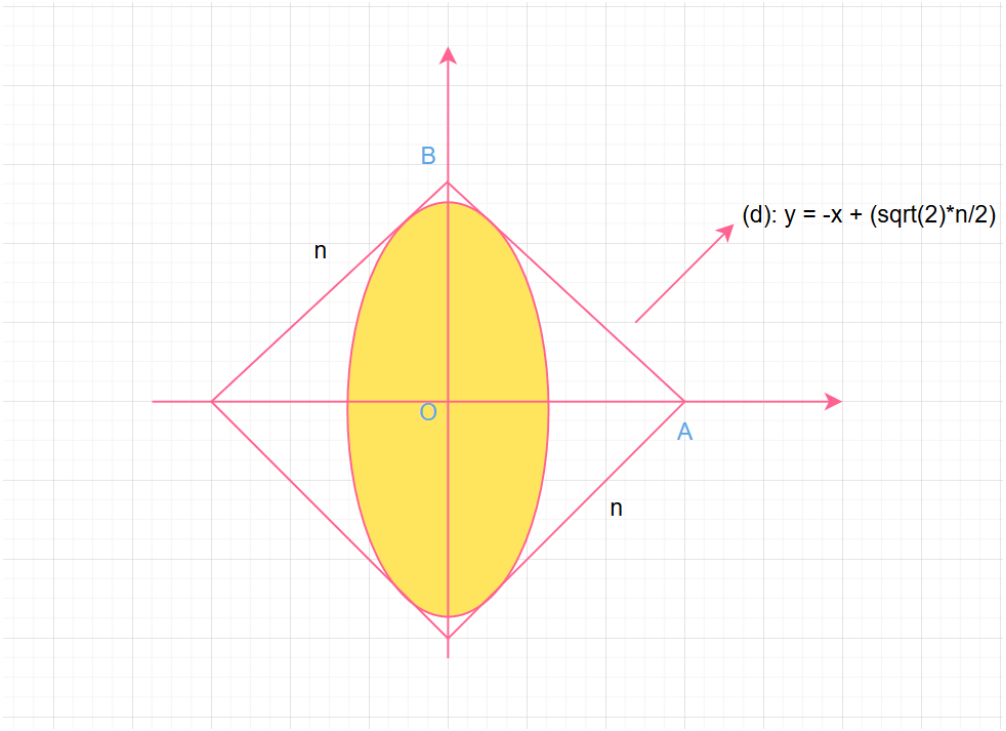
$$\frac{x^2}{a^2} + \frac{y^2}{b^2} \leq 1 \quad \text{(after rotation)}$$

Idea:

The idea is to find a, b based on n, where n is the length of the edge of the image.



For simple explanation, we just work on 1 ellipse, this explanation will work on both ellipses by the way.



Our problem is drawn on Oxy

For the ellipse to be tangent to a side of the square — that is, to intersect it at exactly one point — the system of equations formed by the ellipse and the line representing that side must have exactly one solution. Algebraically, this corresponds to the resulting equation (in x or y) having exactly one real root.

We already have:

- The equation of line AB: $(d) : y = -x + \frac{\sqrt{2}n}{2}$
- The equation of ellipse: $\frac{x^2}{a^2} + \frac{y^2}{b^2} = 1$

The equation of x-coordinate(s) of the intersection:

$$b^2 - \frac{x^2 b^2}{a^2} = x^2 - \sqrt{2}nx + \frac{n^2}{2}$$

$$\Leftrightarrow (\frac{b^2}{a^2} + 1)x^2 - \sqrt{2}nx + \frac{n^2}{2} - b^2 = 0$$

The equation has exactly one solution if and only if:
$\Delta = 0 \Leftrightarrow b^2 - 4ac = 0$

$\Leftrightarrow 2n^2 - 4(\frac{b^2}{a^2} + 1)(\frac{n^2}{2} - b^2) = 0$

$\Leftrightarrow n^2 - (\frac{b^2}{a^2} + 1)(n^2 - 2b^2) = 0$

$\Leftrightarrow n^2 - (\frac{n^2 b^2}{a^2} + n^2 - \frac{2b^4}{a^2} - 2b^2) = 0$

$\Leftrightarrow \frac{2b^4}{a^2} + 2b^2 - \frac{n^2 b^2}{a^2} = 0$

$\Leftrightarrow 2b^4 + 2b^2 a^2 - n^2 b^2 = 0$

$\Leftrightarrow 2b^2 + 2a^2 = n^2$

$\Leftrightarrow a^2 = \frac{n^2}{2} - b^2$

$a > 0 \Rightarrow (0 < b < \frac{\sqrt{2}n}{2}) \Rightarrow (0 < b^2 < \frac{n^2}{2})$
We need a parameter α (alpha) so that we can control the length of b, and we define:

$$b^2 = \alpha \cdot \frac{n^2}{2}$$

$$(0 < \alpha < 1) \Rightarrow (0 < b^2 < \frac{n^2}{2})$$

Input

- `img` (`np.ndarray`): An RGB image represented as a NumPy array with shape `[height, width, 3]`.
- `alpha` (`float`): A scalar in the range $(0, 1)$ that controls the length of the ellipse's semi-minor axis b, defined by $b^2 = \alpha \cdot \frac{n^2}{2}$.

Output

- `masked_img` (`np.ndarray`): The resulting image after applying the elliptical mask. Has the same shape as the input image.

Functionality:

- Defines major and minor axes of the ellipse using the α ratio.
- Applies rotation matrices to transform image coordinates into rotated ellipse frames.
- Combines the masks of both ellipses and overlays them on the original image to preserve only those regions.

3 Results and conclusion

3.1 Results

3.1.1 Original image

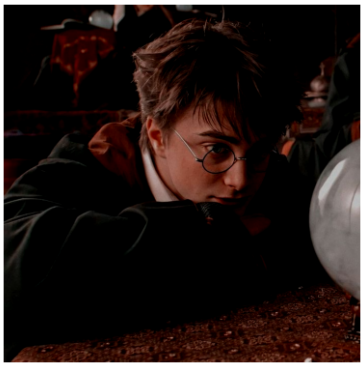
Source: Cutest and Most Adorable Harry Potter Images - 25/7/2025 3:06PM (access time)

Figure 1: The original image (700x700 with 543KB)

3.1.2 Brightness control

Original image

$\alpha = 55$ in 0.01713s



α = -50 in 0.02282s

$\alpha = 100$ in 0.01609s

$\alpha = 200$ in 0.02315s

α = -200 in 0.01667s

Evaluation:

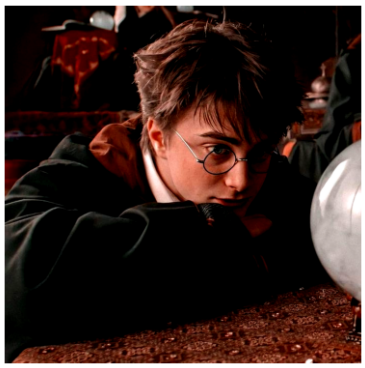
- **Visual Effects:**
  - Positive values of α increase image brightness by shifting pixel intensities upward. For instance:
    - $\alpha = 55$ results in a natural brightness boost with preserved details.
    - $\alpha = 100$ and $\alpha = 200$ cause overexposure, where most image regions become washed out due to pixel value clipping at 255.
  - Negative values of α darken the image:
    - $\alpha = -50$ produces a darker image that still retains meaningful visual information.
    - $\alpha = -200$ leads to extreme underexposure, rendering the image nearly black with most content lost.
- **Performance:**

  - All brightness adjustments were completed within approximately 0.016s to 0.023s.
  - Execution time remains stable and efficient regardless of the α value.

- **Conclusion:**
  - Moderate values of α (e.g., $|\alpha| \leq 55$) provide a good balance between visual clarity and brightness change.
  - Excessive α values may degrade image quality due to saturation (too bright) or loss of detail (too dark).
  - Brightness control via linear transformation is computationally fast and suitable for real-time applications.

3.1.3 Contrast control

Original image



$\alpha = 1.5$ in 0.03368s

$\alpha = 0.1$ in 0.03094s

$\alpha = 0.5$ in 0.03696s

$\alpha = 4$ in 0.03142s

$\alpha = 100$ in 0.03482s

Evaluation

- **Visual Effects:**

  - The contrast control is performed using the transformation output $= \alpha \cdot (\text{input} - 128) + 128$, where α adjusts the intensity spread.
  - When $\alpha = 0.1$ or 0.5, the contrast is heavily reduced, making the image appear faded and flat.
  - $\alpha = 1.5$ enhances contrast moderately, making the image more vibrant while preserving details. It provides a visually pleasing result without overexposure.
  - Larger values such as $\alpha = 4$ or 100 significantly increase contrast but may introduce over-saturation, detail loss, and clipping artifacts — especially noticeable at $\alpha = 100$.

- **Performance:**
  - All transformations are fast, with runtime ranging between 0.0309s and 0.0369s.
  - The processing time does not increase with higher α, showing that the contrast adjustment is computationally inexpensive.

- **Conclusion:**
  - The optimal visual effect is achieved with $\alpha = 1.5$, which enhances contrast naturally without distortions.
  - Very low or high α values lead to undesirable results—either too flat or too harsh.
  - This technique is both visually effective and computationally efficient, making it suitable for real-time applications.

3.1.4 Image flipping

Original image

Image flipped vertically

Image flipped horizontally

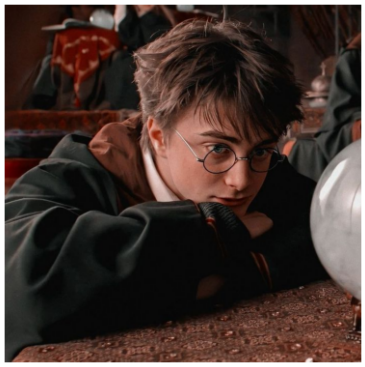
Evaluation

- **Visual Behavior:**
  - Vertical flipping mirrors the image upside down, which may simulate an inverted view or help in rotation-invariant learning.

  - Horizontal flipping mirrors the image left to right, commonly used in training to prevent bias toward one direction.
  - Both transformations preserve the color and structure of the image without distortion.

- **Performance:**
  - The operation is computationally lightweight and fast (0.00009s - 0.00061s).
- **Conclusion:**
  - No loss of image quality, structure, or content.

3.1.5 Color filters



Original image

Grayscale in 0.02192s

Sepia in 0.03259s

Memory evaluation:

harry0_gray

Share

Details

Type	PNG File
Size	356 KB
File location	D:\code\Jupyter Notebook\Pr...
Date modified	7/21/2025 10:38 AM
Dimensions	700 x 700

Properties

Grayscale with 356KB

harry0_sepia

Share

Details

Type	PNG File
Size	548 KB
File location	D:\code\Jupyter Notebook\Pr...
Date modified	7/21/2025 10:38 AM
Dimensions	700 x 700

Sepia with 548KB

Evaluation

- **Grayscale Filter:**
  - Removes color information, preserving only luminance.
  - Helps reduce computational complexity.
  - Helps reduce the storage.
  - Executed in `0.02192s`.
- **Sepia Filter:**
  - Applies a warm brown tone for a vintage effect.
  - Executed in `0.03259s`.

- **Conclusion:**
  - Both filters modify image color while preserving structure.
  - Grayscale is practical for simplification; sepia is artistic.
  - Both are fast enough for real-time or batch processing but sepia costs more space to store.

3.1.6 Image Convolution

Original image

Box blur in 13.8325s

Sharpen in 15.9645s

Evaluation

- **Box blur:**
  - Smooths the image by averaging neighboring pixel values.
  - Reduces noise but also diminishes important details, especially edges and textures.
  - Fast to compute, but suitable for applications where simplicity and speed are required.
- **Sharpen:**
  - Enhances edge contrast and fine details, improving clarity.
  - Can amplify noise in darker or textured regions.
  - More computationally expensive than box blur but useful in situations needing detail enhancement.

3.1.7 Crop Functions

Center Crop

Original image

Center-cropped in 0.00010s

- The center-crop operation successfully isolates the central portion of the original image, focusing on the subject's face.
- The processing time is notably fast at 0.0010 seconds, indicating efficient performance.

Circle Crop

Original image

Circle-cropped in 0.02741s

- The circle-crop operation effectively applies a circular mask to the original image, highlighting the subject's face within a circular boundary.
- The processing time is 0.0274 seconds, which is slightly longer but still reasonably efficient given the additional complexity of the circular mask.

Evaluation

- Both cropping methods preserve the key subject effectively.
- Center-crop is faster (0.0010 s) compared to circle-crop (0.0274 s), likely due to the simpler rectangular cropping process.

Ellipse Crop

$\alpha = 0.1$

$\alpha = 0.2$

$\alpha = 0.3$

$\alpha = 0.4$

$\alpha = 0.5$

$\alpha = 0.6$

$\alpha = 0.7$

$\alpha = 0.8$

$\alpha = 0.9$

Evaluation of Ellipse Crop Operations

- The ellipse crop operation with varying α values (0.1 to 0.9) effectively applies an elliptical mask, with the shape becoming more distinct and visually appealing at extreme values ($\alpha \approx 0.1$ or $\alpha \approx 0.9$).
- At $\alpha = 0.5$, the ellipses transition into circles inscribed within a square, as shown in the image, providing a balanced and aesthetically pleasing framing of the subject.
- A notable symmetry is observed: the ellipse for $\alpha = i$ mirrors the ellipse for $\alpha = 1 - i$ (e.g., $\alpha = 0.1$ resembles $\alpha = 0.9$, and $\alpha = 0.2$ resembles $\alpha = 0.8$), indicating a consistent geometric relationship across the range.
- The transition at $\alpha = 0.5$ serves as a pivotal point, where the elliptical distortion minimizes, enhancing clarity and symmetry, which could be optimal for standardized circular crops.
- This symmetry and the distinct shape variation suggest that α can be strategically chosen to achieve either a focused portrait effect (low α) or a wider contextual view (high α).

4 References

- DYclassroom. How to Convert a Color Image into Sepia Image. url. Access time: 1:24PM 23/7/2025.
- Dynamsoft. Color Space Conversion & Binarization for Image Processing. url. Access time: 1:56PM 23/7/2025.
- Wikipedia. Ellipse. url. Access time: 2:59PM 23/7/2025.
- Wikipedia. Kernel (Image Processing). url. Access time: 2:42PM 23/7/2025.
- freeCodeCamp. Python NumPy Tutorial for Beginners. url. Access time: 8:02AM 21/7/2025.

5 Acknowledgement

This report was prepared with the assistance of ChatGPT, Grok in translating, grammar fixing, docString writing in code.